RESPONSE UNDER 37 CFR 1.116
EXPEDITED PROCEDURE
EXAMINING GROUP 2155

PATENT
Attorney Docket No. 202269
Client Reference No. 39086.02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

RAGHURAMAN et al.

Application No. 09/490,981

Art Unit: 2155

Examiner: Qureshi, Shabana

Filed: January 24, 2000

For: METHOD OF TRACING DATA TRAFFIC ON A NETWORK

DECLARATION OF MELUR K. RAGHURAMAN AND VENKATARAMAN RAMANATHAN UNDER 37 CFR 1.131

U.S. Patent and Trademark Office
220 20th Street S. Customer Window, Mail Stop AF
Crystal Plaza Two, Lobby, Room 1B03
Arlington, VA 22202

Dear Sir:

We, Melur K. Raghuraman and Venkataraman Ramanathan, hereby declare that:

1. We are the inventors of the subject matter disclosed and claimed in U.S. Patent Application No. 09/490,981 ("the present application").

2. We have read and understand the claims 1-22 and 24 set forth in the amendment accompanying this declaration.

3. The subject matter claimed in the present application was invented in this country prior to February 12, 1999 as evidence by Attachments A and B.

In re Appln. of RAGHURAMAN et al.
Application No. 09/490,981

4. Attachment A is a printout of an article entitled, "Network Performance Monitoring in Windows NT," dated bearing a copyright date of 1997 and authored by us. The article describes every element of claim 1-22 and 24, which are the claims presently pending in the application.

5. Attachment B is a web page evidencing that the same article was published in the Proceedings of the Int. CMG Conference, which occurred on December 6-11, 1998.

We further declare that all statements made herein of our own information are true and that all statements made on information or belief are believed to be true; and further that these statements were made with the knowledge that willful false statements are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001, and that such willful false statements may jeopardize the validity of the present application or any patent issuing thereon.

Dated: 7/19/04

Melur K. Raghuraman

Venkataraman Ramanathan

Network Performance Monitoring in Windows NT

Melur Raghuraman & Venkat Ramanathan
Microsoft Corporation

Abstract

Most capacity planning efforts for networks have treated the system as a source generator and focused on frame counts and frame bytes by listening to the wire. Some have employed smart ways of identifying the application responsible for network traffic by scanning the packet headers. These methods are expensive and arbitrary. In this paper, we present trace instrumentation of the TCP/IP stack in Windows NT that provides key information for capacity planners for correctly charging network traffic to the individual services and applications.

Introduction

In this client/server world of computing that we live in today, capacity planning activity is just not limited to the server but to the entire system including the clients, servers and network devices. In the past, network capacity planners have treated the systems without serious concern and system capacity planners have in turn ignored the network devices in their analysis. One major reason for this is the lack of appropriate performance data to tie the two worlds and the fear of overhead of instrumentation.

Accuracy of model input metrics is often cited as the key indicator of the validity of a capacity analysis. In the last couple of years, there have been several papers in this conference mentioning the lack of performance data in Microsoft® Windows NT® operating system mainly for Capacity Planning [1,2]. We addressed these concerns by introducing an Event Tracing Facility in Windows NT 5.0 and published the event tracing API[3] in this conference last year. While it addressed system data requirements adequately, data related to network was lacking. The purpose of this paper is to describe the network instrumentation work that has been done to complement the kernel trace instrumentation work for Windows NT 5.0.

With internet access becoming common place today, there is no drop in the appetite for network bandwidth for web based applications. In fact, high speed networking is a very important focus of Windows NT 5.0 which already achieves 1 to 2 Gbps throughput. With network speeds getting this fast, any instrumentation must be highly optimized to take minimal overhead.

In order to place the network instrumentation in the larger context of Networking layers in Windows NT, we devote the next section to describe the networking architecture in Windows NT. We describe the capacity planning data requirements for Networks and what data is collected through the instrumentation. Sample trace data from our experiments are also presented.

Event Tracing in Windows NT 5.0

The next version of Microsoft Windows NT operating system will have a uniform framework for event tracing, specifically for capacity planning. The event tracing mechanism implements a circular buffer pool maintained by the operating system and an API set for Trace Providers, Consumers and Management Control. The trace logger can accept data from kernel mode device drivers and user mode applications. In addition to providing a facility to log trace data for applications, the Windows NT kernel has been instrumented to provide key capacity planning metrics that were not available through the commonly used performance tool 'Perfmon'. The following system events are instrumented:

1. *Process Creation/Deletion* event. The ProcessId, parent process Id, Security Id and the Image File name are recorded.
2. *Thread Creation/Deletion* event. The Thread Id and its process Id are also recorded.
3. *Hard Page fault* event. The disk signature and the size of the first disk read resulting from the page fault are also recorded.
4. *Disk Read/Write* event. The disk signature and the size of the operation are recorded.

"ATTACHMENT A"

Multiple logger streams may be active at one time, typically one for the kernel logger and one for each of the trace-enabled applications running on a server. The Consumer APIset makes it easy to process the trace from multiple logger streams in the proper order and returned to the caller one event at a time.

Networking Architecture in Windows NT

Networking capabilities are built into Windows NT and it is organized as layers (See Figure 1).

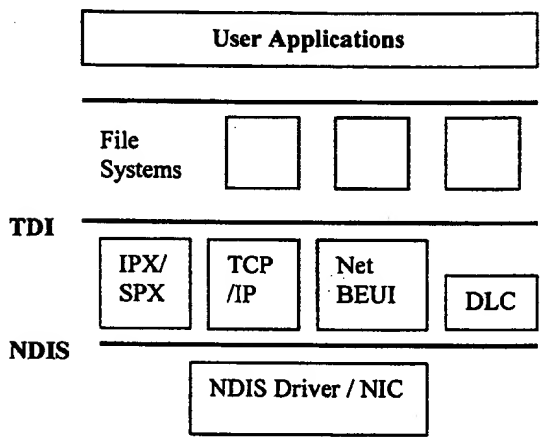


Figure 1. Windows NT Network Layers

Windows NT networking components include:

- Transport protocols (DLC, NetBEUI, NWLink, and TCP/IP) define the rules governing communications between two computers.
- Inter-process communication (IPC) components, such as named pipes and mail slots, allow applications to communicate with each other over a network.
- File and Print sharing components allow resources to be made available on a network.

The Multiple uniform naming convention (UNC) Provider (MUP) and Multi-Provider Router (MPR) make it possible to write applications that use a single API to communicate using any network vendor's redirector.

There are two boundary layers in the architecture, namely the Network Driver Interface Specification (NDIS) and Transport Driver Interface (TDI). The NDIS layer provides the interface and a wrapper to the Network Interface Card (NIC) device drivers. The TDI boundary layer provides a common interface specification to communicate with various transport drivers. While several protocols are supported in the transport layer, TCP/IP forms the main focal point for all networking activity.

The protocol suite benefits from years of research and is the most favored suite in the Internet. In WindowsNT, several services make use of TCP/IP stack, most notably File/Print services and Socket-based applications. Most services require reliable data transmission and use TCP/IP suite for end to end reliable delivery. We will explain the File/Print services and Sockets in more detail in the next section.

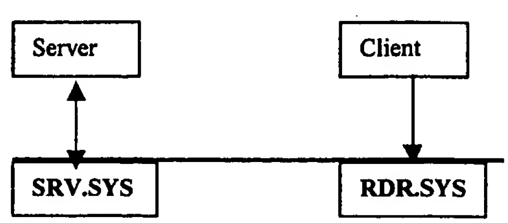
File and Print Services

The File and Print services are supported by two services (Redirector and Server) that are layered on top of Transport Driver Interface layer as shown in Figure 2. They provide an encapsulation over the file system and network transparency for applications accessing remote files.

top of transport

When a process on a Windows NT system tries to open a file that resides on a remote computer, the following steps occur:

- The process calls the I/O Manager to request that the file be opened.
- The I/O Manager recognizes that the request is for a file on a remote computer, so it passes it to the redirector file system driver (RDR.Sys).
- The redirector passes the request to lower-level network drivers that transmit it to the remote Server for processing.
- The transport receives a send IO request packet (Irp) for the SMB header and command.
- This send is translated into frames and queued for dispatch on the wire through the Miniport driver.



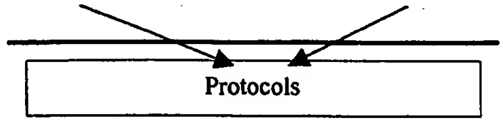


Figure 2. Server and Workstation Services

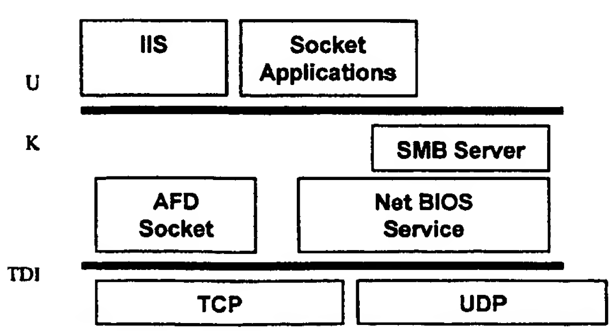
On the remote WindowsNT server system, when the server service receives a request from a remote computer asking it to read a file that resides on the hard disk, the following steps occur:

- The low-level network drivers receive the request and pass it to the server driver (SRV.SYS).
- The server passes a file read request to the appropriate local file system driver.
- The local file system driver calls lower-level disk device drivers to access the file.
- The data is passed back to the local file system driver.
- The local file system driver passes the data back to the server.
- The server passes the data to the lower-level network drivers for transmission back to the client computer.

Socket-based Applications

Socket-based applications are supported through the Windows Socket Provider (WinSock). Figure 3 shows the relationship between various modules and TCP/IP.

A Socket based user application that would like to provide a service on port P (pre-advertised) to all clients would open up a socket through WinSock and listen on the socket for connection requests. This would translate as an address object with TCB structures listening on that port with the server's IP address(es) and a wild-card IP address to denote client addresses.



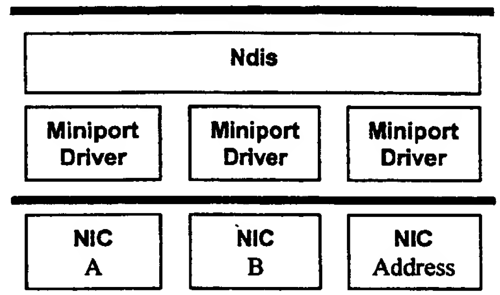


Figure 3. TCP/IP Stack

When a client requests a connection, a frame comes in on one of the NICs (with the client's IP address and connecting port #) and is tied to one of these TCB structures. TCP calls across TDI to indicate to the socket provider, which in turn calls into the User's service application for acceptance. Once accepted, frames are received and sent on the NIC involved in the connection. Though the physical NIC through which the connection data is routed can change over time, the IP addresses and port numbers don't change and lend themselves as connection context for event tracing.

The user application requests a Send to the socket service provider with a pointer to the data. The socket provider, namely AFD will lock the pages in memory and request TCP across TDI to send. TCP cuts the requests to frames and calls the miniport driver corresponding to the NIC through which the data needs to be transmitted. The Miniport driver sends the frames and calls to TCP to complete each frame-send event. The TCP requests are processed asynchronously as a rule, and could happen in the context of the system thread dispatched by the socket provider or a DPC (Deferred Procedure Call) from Ndis.

In the case of receive, the miniport driver services the interrupt and through Ndis, queues a DPC to process the frame. This DPC identifies the protocol stack and calls the appropriate Receive Event handler. When this thread executes TCP Receive routine in its context, the data is indicated up or is filled in pre-posted receive buffers from the application.

Event Tracing Sends / Receives

A TCP Send needs to be traced at the end of the send. The end of a send is marked by the processing of an ACK(acknowledgement) from the other connection endpoint corresponding to the last byte of the send. Through TDI, TCP receives an IoRequestPacket (IRP) from AFD with a pointer to a locked user buffer / locked set of pages from a file's cached view. TCP creates a Send-Request structure, which caches this IRP, splits the data into frames, and sends them across. When the last ack (ack acknowledging the last byte sent) arrives, part of receive processing involves queuing the corresponding Send-Request for completion. When the completion queue is processed, the cached IRP is completed to the upper layer.

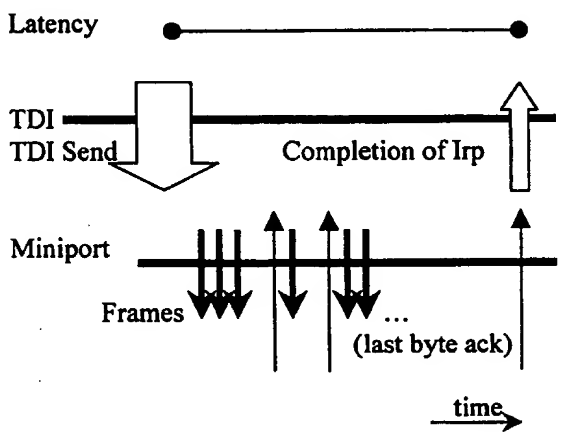


Figure 4. TCP Send

Between the initiation of the Send and the completion, several copies of the data could get transmitted due to retransmissions, or the send could get cancelled, in which case, a completion doesn't occur. In Fig 4., the timeline of events during a send is explained. Since only the completion is traced, the use of the NIC to send out the said number of bytes is guaranteed. Also, as far as the user application is concerned, the Send through Tdi completed only when the IRP is completed in the socket driver (which may wake up the blocked thread in the case of synchronous i/o or trigger the appropriate event in the case of async i/o).

Tracing receives is more complex than Sends, in the sense that tracing information needs to be generated at more points than one. In the case of receive, we should not care if the TCP protocol sends out acks or if this is only part of a receive which got cancelled from the other end point. The number of bytes received must be accounted for exactly. We will first take the case of a pre-posted receive and how it is traced for capacity planning purposes.

In the case of pre-posted receives, TCP receives an IRP through TDI to receive a certain size. TCP caches this IRP in a Receive-Request structure. When a chunk of a certain size of data is received (could be less than the requested size, to improve latency) TCP completes the request with the then-available number of bytes and the appropriate buffers. If more needs to be received, the receiver (say application through socket interface) posts more receive-IRPs which are completed as frames are received. In Fig 5., the receive completion and trace timing is explained.

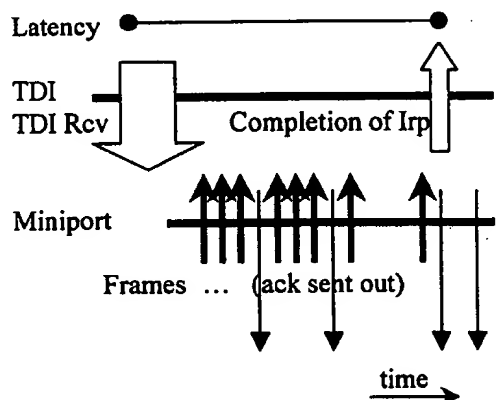


Figure 5. TCP Receive

It is possible to receive when no receives are posted. In such cases, the data is indicated to the receiver as soon as the first frame is received. If any more data needs to be received, TCP receives a piggybacked Irp. TCP generates a CP trace in this indicate path to accurately account for the accepted number of bytes.

Data Collection Issues

Windows NT uses a packet oriented I/O model for performing I/O operations to disks as well as to network devices. Whenever a user application or service posts a Send/Receive request, an IRP is created and sent to TCP. Typically the IRP is filled with the context of the thread requesting the operation. In the case of sends, it is possible to identify the correct user thread to charge the bandwidth utilization. In the case of receives, a DPC is generated in NDIS, which doesn't run in the context of any user thread. With respect to receives through the indicate-path described

above, when the data is accepted without requesting more receives through IRPs, it is not possible to make a correspondence. In such cases however, the port information that is provided is useful in identifying the service.

What is Instrumented?

1. *Send Complete* event when a TCP send request is complete (when an ACK for the last byte of the Send Request is received). The source address, destination address, source port number, destination port number, bytes transmitted are also recorded. Events are automatically time-stamped by the trace logger.
2. *Receive Indicate* event when incoming data is indicated to the upper layers. The source address, destination address, source port and destination port numbers, the size of data received and the process Id of the Process that is being indicated by TCP.
3. *Receive Complete* event (when a receive-Irp is completed with data). Similar information is collected.

These three trace points cover the majority of the meaningful TCP traffic in the system. It is important to keep in mind that some TCP traffic is not accounted for by this instrumentation. For example, retransmissions from packet loss, receiving IP control msgs (like ICMP etc.).

Instrumentation Overhead

In comparison to other kernel events such as thread create or delete, network events are very high frequency events. As a result, extraordinary care has been taken to minimize the overhead of trace instrumentation. The data being collected is primarily from the Transport Control Block (TCB) structure. The fields in the TCB structure are arranged to make the data relevant to capacity planning in one contiguous block. This allows direct copying from the TCB structure to the Trace Buffers without having to make any intermediary copies. In our measurements, a network event uses about 128 x86 instructions and logs 24 bytes of data.

Analysis of sample Traces

Appendix A provides a sample kernel trace fragment from a TCP Send test, translated into readable text format. Each row in the table shows an event instance. Each event instance is described by the fixed header providing the event name, Thread ID that is causing the event, system clock time when the event happened, the kernel and user mode CPU time for the thread. Additional columns in the table show the event specific data associated with each event.

The tests were started after starting up the Trace logger using a command line utility called tracelog.exe. The trace shows the process start and end of the tracelog.exe. Next we see a process start for the TCP send test program (nttcp.exe). Immediately following that we see the Tcp Send events. We can see the source IP address/port and destination IP address/port.

The size of transfers is 8K bytes. The thread that's actually performing the send is (thread Id 0, a system thread). However, the process Id that was saved when the connection was created is 1C0, recorded with every send event. This process Id 1C0 corresponds to the nttcp.exe program that initiated the sends. Hence, we can see that the network traffic can be charged to processes properly from our traces.

While the TCPSend events triggered on the nttcp connection were in progress, an HTTP request for a webpage happened (shown in italics) and through threadID 39C, this request was handled by the IIS running on port 80 and 3 files were sent out to the HTTP remote browser. The HTTP transaction happened through a keep-alive connection. These events were charged to process 382 (InetInfo.exe).

Possible uses of network traces

Classification by PID:

In summary, since stack event trace generated incorporates PID, it is possible to charge network traffic to a specific process or kernel mode service. We will present other possible modes of classification, such as per-service, per-NIC, per-remote-request or per-client as indicated in paragraphs below.

Classification of network utilization per Service:

From post-processing the collected trace information, it is possible to classify the

bandwidth utilization by application / service. Services and user applications / connections can be characterized using the 4-tuple (SAddr, DAddr, Sport, Dport). The traces collected for the specific port show the utilization for a particular service. From the traces shown, it can be observed that (HTTP) web service active on port 80 can be charged for the receives and sends in italics.

Classification of network utilization per NIC:

Using tools such as IPConfig, it is possible to identify NICs and assigned IP addresses. Parsing the collected trace for a specific IP Source Address gives all the traffic for that particular NIC. It is possible that data rerouting can happen when a transfer is in progress. In such a case, TCP receives an indication and a special stack event trace is generated.

Classification of System resource utilization per remote request / client:

Since Disk I/O events are generated in the context of the process, and remote requests are charged to the same process (through stack event traces), it is possible to identify the running service to charge disk i/o operations to. Based on Destination IP addr, this can be further classified per client of the service.

Conclusion

With the introduction of Event Tracing for Windows NT 5.0, we can charge resource consumption of CPU, Memory (Page faults), Disk I/O and Network to applications or Services. This will make the task of capacity planning client/server applications running on Windows NT servers easier and more accurate.

References

1. *Beyond Bandwidth – Mainframe Style Capacity Planning for Networks and Windows NT,* J.P. Buzen and A. W. Shum, Conference Proceedings CMG 96.
2. *Considerations for Modeling Windows NT,* J. P. Buzen and A. W. Shum, Conference Proceeedings CMG 97.
3. *Capacity Planning for Windows NT: Event Tracing and Instrumentation,* Jee Fung Pang, CMG 97 Late Breaking Paper.
4. *Inside Windows NT (Second Edition),* David Solomon, Microsoft Press 1998.
5. *Networking in Microsoft Windows NT,* Glen Clark, MSDN Technical Articles.

Appendix A:

Event Name	TID	Clock (ms)	Kt	Ut	Parent TID	Parent PID	Sz	Image Name		
ProcessStart	1C0	1124570445	2	0	10C	34C		Tracelog.exe		
ThreadStart	1C0	1124570445	2	0	1C0	10C				
ThreadEnd	1C0	1124570492	3	0	1C0	10C				
ProcessEnd	1C0	1124570492	3	0	10C	34C		Tracelog.exe		
DiskWritelo	14	1124571070	0	0	2	67	300			
ProcessStart	2AC	1124572773	36	28	1C0	288		CMD.exe		
ThreadStart	2AC	1124572773	36	28	10C	1C0				
ThreadStart	10C	1124572898	0	2	364	1C0				
Event Name	**TID**	**Clock (ms)**	**Kt**	**Ut**	**Source Addr**	**Dest Addr**	**Sport**	**DPort**	**Size**	**PID**
TcpSend	0	1124572976	26425	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124572976	26425	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124572976	26425	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124572992	26426	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124572992	26426	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573008	26426	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573008	26426	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573023	26427	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpRecv	***0***	***1124573093***	***78293***	***1***	***172.31.249.34***	***172.31.254.11***	***80***	***6437***	***531***	***382***
TcpRecv	***0***	***1124573164***	***78294***	***3***	***172.31.249.34***	***172.31.254.11***	***80***	***6437***	***164***	***382***
TcpSend	***39C***	***1124573198***	***78296***	***0***	***172.31.249.34***	***172.31.254.11***	***80***	***6437***	***1507***	***382***
TcpSend	***39C***	***1124573231***	***78303***	***0***	***172.31.249.34***	***172.31.254.11***	***80***	***6437***	***9236***	***382***
TcpSend	***39C***	***1124573362***	***78304***	***0***	***172.31.249.34***	***172.31.254.11***	***80***	***6437***	***6728***	***382***
TcpSend	0	1124573570	26460	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573586	26461	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573586	26461	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573601	26462	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573601	26462	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573617	26463	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573617	26463	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573633	26464	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573648	26465	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
TcpSend	0	1124573648	26465	0	172.31.249.34	172.31.255.147	5010	5020	8192	1C0
Event Name	**TID**	**Clock (ms)**	**Kt**	**Ut**	**Parent TID**	**Parent PID**	**Sz**	**Image Name**		
ThreadEnd	364	1124573758	0	0	364	1C0				
ThreadEnd	10C	1124573789	1	3	10C	1C0				
ProcessEnd	10C	1124573789	1	3	1C0	288		nttcp.exe		

Legend: Kt - Kernel Mode Time
Ut - User Mode CPU Time
PID - Process ID
TID - Thread ID

The following are the extended record fields for the respective events:

- Process start - new Process Id, its Parent's Process id
- Process end - current Process Id, its Parent's Process id, the image filename that it was running
- Thread start - new thread Id, its Process Id
- Thread end - current thread Id being terminated, its Process Id.
- I/O read, I/O write - The signature of the disk where the I/O operation was done, the transfer size.
- TCPSend - Source Address, Destination Address, Source port, Destination port, Size, ProcessId

---

dblp.uni-trier.de

24. Int. CMG Conference 1998

24th International Computer Measurement Group Conference, December 6-11, 1998, Anaheim, California, USA, Proceedings. Computer Measurement Group 1998

```
@proceedings{DBLP:conf/cmg/1998,
  title     = {24th International Computer Measurement Group Conference, December
               6-11, 1998, Anaheim, California, USA, Proceedings},
  booktitle = {Int. CMG Conference},
  publisher = {Computer Measurement Group},
  year      = {1998},
  bibsource = {DBLP, http://dblp.uni-trier.de}
}
```

Tuesday, December 8, 1998

Session 1

- Yi-Chun Chu: Performance Measurement of the PeopleSoft Multi-Tier Remote Computing Application. 1-25
- Thomas Beretvas: DASD Tuning in the New DASD Environment. 26-38
- Gregory V. Caliri: Getting Started in Mainframe Capacity Planning. 39-43
- Melur K. Raghuraman, Venkat Ramanathan: Network Performance Monitoring in Windows NT. 44-50
- Joel Goldstein: DB2 Buffer Pool Tuning - Top Down or Bottom Up? 51-62
- Brian L. Wong: Comparing MVS and UNIX Workload Characteristics. 63-69
- Steven E. Smead: Service Level Instrumentation 101 - An In-Depth Look at How to Instrument End User Transactions. 70-84
- Daniel P. Ruehl: Business Centric Resource Management Reporting. 85-93

Session 2

- Charles L. Burmaster, Robert Daniel Johnson, Robert H. Johnson: Performance Metric Validation in Chaos. 94-105
- Bruce McNutt: Disk Capacity, Performance, and Cost: Maintaining the Balance. 106-113
- Tim Follen: Capacity Planning and Performance: Tips and Techniques. 114-120
- Mike Tsykin, Christopher D. Langshaw: End-To-End Response Time and Beyond : Direct Measurement of Service Levels. 121-132
- Bernard Domanski: NT Benchmarking. 133-149
- Joseph L. Hellerstein, Fan Zhang, Perwez Shahabuddin: Characterizing Normal Operation of a Web Server: Application to Workload Forecasting and Problem Determination. 150-160
- Ted McGavin: Fibre Channel: The Third Age of Disk Connectivity. 161-165

Session 4

"ATTACHMENT B"

Session 5

Wednesday, December 9, 1998

Session 1

Session 2

- Thomas E. Bell, Russell C. Davis: Scrambling To Upgrade Infrastructure Software For Year 2000. 422-430
- Doug V. Johnson: Database Layout Guidelines for Large Queries. 431-436
- Sudhir R. Nath: Capacity Planning and Performance Tuning of the UNIX Systems. 437-447
- Daniel A. Menascé, Nagesh Kakarlamudi: Performance and Availability of Rplicated Database Servers. 448-455

Session 4

- Alan Harbitter, Daniel A. Menascé: Performance Issues in Large Distributed System Security. 456-467
- George W. Dodson: IT Availability - It's Critical to Your Business. 468-487
- Danilo Gotta Luca Biffi: Performance Evaluation of Web Applications. 468-
- Richard S. Ralston: A Unique VTS Experience. 488-495
- Hari Sivaraman, Swami Ramany: Scalability Analysis of SCSI Connected IO Systems. 496-506
- Luca Pagliarini, Silvio Pastore Stocchi: From Design to Delivery: A Practical Experience on Performance Prediction. 507-517

Session 5

- Anthony G. Mungal: I/O Concurrency, I/O Bandwidth, Access Density and the Like. 518-527
- Mark W. Johnson: Measuring Service Levels in E-business Applications and Applets. 528-538
- Marco A. Mendes, Virgilio Almeida: Analyzing the Impact of Dynamic Pages on the Performance of Web Servers. 539-547
- Frank M. Bereznay: Who Needs Virtual Tape When You Have Fat DASD? Maybe You Do! 548-555
- Edward A. Woods: Expanding The Reach Of DB2: TCP/IP and DRDA. 556-565

Thursday, December 10, 1998

Session 1

- Irwin F. Kraus: The RMF Type 70 Record: A Plethora of Possibilities. 566-576
- Joseph K. Merton: Organizational Behavior to Systems Implementation Failures. 577-581
- Wayne A. Wyrobek: Tales from the Crypto: Performance Alerts for Mainframe Users of Integrated Cryptographic Facilities. 582-592
- Don O. Koch, Gail Ridgley, Carl Meredith: Using Unix Accounting Utilities for Chargeback Accounting. 593-601
- Harry J. Foxwell, Daniel A. Menascé: Prefetching Results of Web Searches. 602-609
- Jee Fung Pang, Melur K. Raghuraman: Understanding the Windows NT I/O Subsystem. 610-616

Session 2

- H. Pat Artis: DIBs, Data Buffers, and Distance: Understanding the Performance Characteristics of ESCON Links. 617-625
- Jack B. Woolley: Creating a Response Time Agreement for C/S Applications is a Whole New Ballgame! 626-635
- Jodi E. Perry: ARM Yourself for Continuous Availability. 636-642
- Jim Skeen: Data Warehouse Performance: A Methodology for Analysis. 643-654

Session 4

Session 5

Friday, December 11, 1998

Session 1

Session 2
